J.記号計算 (1998) **26**, 89-95 論文番号
sy980202

順列群における因数分解問題を解くアルゴリズム

トルステン・ミンクヴィッツ

ドイツテレコム（ドイツ、ボン

順列群における因数分解問題とは、ある順列群Gの要素gを、与えられたGの生成子集合S上の単語として表現する問題である。計算群論の他の多くの問題と同様に、この問題はGの強生成集合(SGS)と基底から解くことができる。古典的なアルゴリズムはSchreier-Sims法である。しかし、因数分解のためには、すべての要素がS上の単語として表現されたSGSが必要である。本稿では、因数分解問題を解くための簡単なアルゴリズムを紹介する。これは、生成子上の比較的短い単語で表現された要素を持つSGSを計算することに基づく。

1. はじめに

1980年代初頭に流行したゲームにルービックキューブがある。ルービックキューブは6つの面を持つ立方体で、それぞれ色が異なり、90°ステップで手で回すことができる。その仕組みは、各面を3×3の小さな正方形に細分化する必要がある。ほんの数回ランダムに回すだけで、各面にさまざまな色を混ぜることができる。このゲームの目的は、立方体を元の一色の面に戻すことである（このゲームについての詳しい説明はHofstadter (1985)を参照）。

この記事の目的にとって興味深いのは、立方体の面を1回転させるごとに、6×9＝54個の小さな正方形が並べ替えられるという事実である（中央の正方形はそのままなので、回転によって移動するのは48個だけだが）。6つの異なる90°ターンは順列群の生成子である。このゲームを解くには、与えられた立方体の状態に対応するグループ要素の逆数を因数分解しなければならない。本稿で紹介するアルゴリズムは、ルービックキューブゲームを解くものである。しかし、これは順列群における因数分解問題を解くための一般的な方法である。それにもかかわらず、キューブはこの研究のインスピレーションとなっている。

Gを有限集合Ωの双射の集合Sによって生成される群とする。そして、$x\in\Omega$に対するGの部分群$\mathrm{Stab}(G, x) := \{g\in G \mid x^g = x\}$をxのスタビライザーと呼び、$Gx := \{x^g \mid$

$g \in G\}$ をxのG軌道と呼ぶ。Ωは任意の、しかし固定された方 法 で 並べられる：Ω $= \{x_1, \ldots, x_n\}$とする。部分群

$$G^{(i)} := \{g \in G \mid \forall_{1 \leq j < i} : x_j^g = x_j\}. \tag{1.1}$$

のGをi番目の安定化子と呼ぶ（したがって$G = G^{(1)}$ と $\mathrm{Stab}(G, x_1) = G^{(2)}$）。

数列

0747-7171/98/070089 + 07 $30.00/0

すべての $G^{(i)}$ のうち、$1 \le i \le n$ を安定化鎖と呼ぶ。強生成集合(SGS)はGの部分集合Rであり、すべての$1 \le i < n$に対して $\langle R \cap G^{(i)} \rangle = G^{(i)}$ という方程式が成り立つようなものである。すべての$x \in B$に対して$x^g = x$となるようなすべての$g \in G$が恒等式である場合、Ωの部分集合BはGの基底と呼ばれる。明らかに、Ωは基底である。これらの定義についてのより完全な議論はButler (1991)にある。

Let $B = \{b_1, \ldots, b_k\} \subset \Omega$ be a fixed and ordered base of G and Ω be ordered in such a way that $b_i = x_i$, $1 \le i \le k$. Then partial maps v_i, such that

$$v_i : \Omega \to G^{(i)}, \quad \text{そして} \quad \begin{array}{ll} \omega^{v_i(\omega)} = b_i & \omega \in G\, b^{(i)}_i \\ v_i(\omega) & \text{未定義 else} \end{array} \tag{1.2}$$

$^{(i+1)(i)}$これは、表の定義された項目（したがってコセット）は、b_i の$G^{(i)}$ -軌道の要素によってインデックス付けされていることを意味する。

$^{(1)}{}_1{}^{(2)}g$ がb_1 を ω_1 に写すなら、 $gv_1(\omega_1)$は $G^{(2)}$ にある。$^{(3)}gv_1(\omega_1)$ がb_2 を ω_2 に写すなら、 $gv_1(\omega_1)v_2(\omega_2)$ は $G^{(3)}$ にある。これを繰り返すと、最終的に $G^{(k+1)} = \langle_G$ に到達する。

$$gv_1(\omega_1)v_2(\omega_2) \cdots v_k(\omega_k) = 1_G . \tag{1.3}$$

このように、順序付きベースBとすべてのテーブルv_i が与えられたとき、任意の$g \in G$は一意な因数分解を持つ。

$$g = v_k(\omega)_k^{-1}\, v_{k-1}(\omega)_{k-1}^{-1} \cdots v_1(\omega)_1^{-1} . \tag{1.4}$$

したがって、v_i のイメージがすべて生成子S上の単語として表現されていれば、因数分解の問題は解決する。表から関連する単語を反転し、結果を連結するだけである。

集合$R := \bigcup_{1 \le i \le k} \mathrm{Image}(v_i)$ は常にGのSGSであり、Gの次数は表サイズの積で計算できる。この性質はアルゴリズムの終了基準となる：

$$\mathtt{TableFull}(G, \{v_i\}_i) := |G| = \prod_{1 \le i \le k} |\mathrm{Image}(v_i)| . \tag{1.5}$$

欠けているのは、順列群Gの与えられた生成子S上のすべてのエントリを単語として表現したテーブルv_i を計算するアルゴリズムである。

2. アルゴリズム

そのために、アルゴリズム中に使用されるグループ要素は、順列として、またS上の単語として表現される。しかし、必要な演算は反転と乗算の2つだけである。したがって、これは難しくない。アルゴリズムの基本的な考え方は非常に単純である：式(1.2)を定義する表をチェックするために、多くの群要素を使用する。必要な表

~~項目がまだわからない場合は、それを見つけるのに失敗したグループ要素を~~

使用する。詳細には、ある$i \in \{1 \ldots k\}$と$t \in G^{(i)}$ の場合である：

$v_i(b^t)_i$が定義されている場合

では

$r := tv_i(b_i^t)$；

```
その他
  set vi (bti ) to
  t−1 ; r := 1G ;
Fiだ;
```

このIf句の後条件はr（$G^{(i+1)}$）である。したがって、tをrに設定し、iを1増やして再び句を適用することができる。これをi = |B|になるまで繰り返す。この単純な手順を、短い単語を持つ要素$t \in G^{(1)} = G$の列で何度も繰り返す。この過程で表は埋められていく。簡単な方法は、まずS上の単語長1の要素をすべて使用し、次に単語長2の要素をすべて使用し、そのように繰り返すことである。新しいtを1つ提供することをラウンドと呼ぶ。手続きNext()は、次のラウンドのtを返す。Next()のパラメータはSとラウンドのカウントである。すべてのテーブルが満杯になり（式(1.5)参照）、あらかじめ定義されたラウンド数nに達すると、いつでも停止することができる。

明らかな改善点がいくつかある：

1. tが現在の$v_i(b^t_i)$よりも短い単語を持つ場合、$v_i(b^t_i)$をt^{-1}に設定することができる。はワード長を一定に保つ。(実際には、2つのワード長が等しいときに設定するのが良いアイデアでさえある。なぜなら、結果として生じるジッターが、v_iが悪い状態で「立ち往生」するのを防ぐからである)
2. t^{-1}が$v_i(\omega_i)$の既知の最短候補である場合、tは多くの場合良い候補である。for $v_i(b^{t^{-1}}_i)$.
3. 各ラウンドの計算は、tがある限界値lを超えるワード長を持つか、$t = 1_G$を超えるたびに新しいラウンドが開始される場合、大幅に高速化される。

このように、If句は再帰的手続きとなるように変更される。チルダ~は、参照渡しされ、変更可能なパラメータを示す。

```
Step := Procedure(G, B, i, t, ~ r, ~ {v }jj )
  vi (bt ) iが 定義されている場合。
    では
      r := tvi (bti );
      Wordlength(t) < Wordlength(vi (bt )) iの場合。
        では
          vi (bt ) iを t−1 にセットする;
          Step(G, B, i, t−1 , ~ r′ , ~ {v }jj );
      Fiだ;
    その他
      vi (bt ) iを t−1 にセットする;
      Step(G, B, i, t−1 , ~ r′ , ~ {v }jj );
      r := 1G ;
  Fiだ;
```

レベルiから始まるラウンドが指定される：

```
Round := Procedure(G, B, l, c, ~ {v }jj , ~
  t) i := c;
  While (t /= 1G ) and (Wordlength(t) < l)
    Do Step(G, B, i, t, ~ r, ~ {v }jj );
```

```
    t := r;
    i := i + 1;
  オッド
```

これらの改良の結果生まれたアルゴリズムは、すでにかなりの性能を発揮している：

```
SGSWord := Procedure(G, S, B, l, n, ~ {v})ii
  vi (bi) = 1G を除き、すべてのvi をあらゆる場所で不定とする;
  count := 0;
  While (count < n) or (not TableFull(G, {v}ii ))Do
    t := Next(S, count);
    count := count + 1;
    Round(G, B, l, 1, ~ {v}ii , ~ t);
  Od;
```

しかし、まだ多くの欠陥がある。非常に重要なのは、v_i の新しく良い（短い単語を意味する）画像を見つけたにもかかわらず、それを十分に利用しないことによる無駄である。これは、sラウンドごとに停止し、各iのv_i の新しいエントリを見ることで補うことができる。これらは他のエントリと乗算され、その結果がレベルiのRound()で使用する新しいtとなる。これは多くの場合、v_i のイメージを改善する。sの良い選択は難しいが、ベース｜B｜^{2} のサイズの2乗は通常悪くない。

それでもなお、このアルゴリズムがかなり悪い結果を出す場合もある。顕著な例は、対称群S_N である。

$$s_N := \langle (1, 2), (2, 3), (3, 4), \ldots, (n - 2, n - 1), (n - 1, n)\rangle. \tag{2.1}$$

ジェネレーターは非常に多いので、十分な語長のtを持つラウンドに到達するには長い時間がかかる。各GとSについて、tがすべてのレベルのすべてのコセットをカバーするためには、特定の単語の長さが必要である。また、SGSWord()自体も悪いフィニッシャーである。最後の数エントリを見つけるのに長い時間がかかるかもしれない。これらの問題の解決策は、すべてのテーブルを埋めることに特化した別の手続きを sラウンドごとに実行することである（以下の手続きFillOrbits()を参照）。これは非常に単純なアルゴリズムで、全テーブルの全エントリーを調べ、各レベルiについて、これまで未定義であったv_i (ω)のエントリーを見つけようとするものである。

パラメータlは、最初のうちは小さく設定すべきである。テーブルがまだ一杯でない間は、徐々に大きくすることができる。lの値を小さくすると、ラウンドが早く終了するため、各ラウンドの実行時間が短縮される。完全なアルゴリズムは、以下の擬似コード手順で指定される：

```
SGSWordQuick := Procedure(G, S, B, n, s, ~ l, ~ {v}ii )
  set all vi undefined anywhere, except vi (bi) = 1G ; count :=
  0;
  While (count < n) or (not TableFull(G, {v}ii ))Do
```

```
t := Next(S, count);
count := count + 1;
Round(G, B, l, 1, ~ {v}ii , ~ t);
If (count mod s) = 0
  では
    Improve(G, B, l, ~ {v}ii );
```

```
    If (not TableFull(G, {v }ii ))
      ならば
        FillOrbits(G, B, l,~ {v }ii );
        l :=5 l;4
    Fi
  Fi だ
  だ
  ;   ;
オッ
ド

Improve := Procedure(G, B, l,~ {v }ii )
  For j From 1 To |B| Do
    For x In Image(vj ) Do
      For y In Image(vj )
      Do
        もし（xかyがImage(vj )で新しい
          ）なら、次のようになる。
            t := x y;
            Round(G, B, l, j, ~ vi , ~ t);
        Fiだ;
      オッド
    オッド
  オッド

FillOrbits := Procedure(G, B, l,~ {v }ii )
  For i From 1 To |B| Do
    O := {by i: y∈Image(vi )}; (部分軌道は発見済み) For x InUi<j≤|B|
    Image(vj )Do
      For p In Ox - O Do（軌道の新しい点を歩く）
        t := v i(px-1)x;
        もしWordLength(t) < lなら
          では
            vi (p) を t−1 にセットする;
        Fiだ;
      オッド
    オッド
  オッド
```

結果として得られる表の品質は、因数分 解 の 最大単語長で測定される。これは

$$\max_{1 \leq i \leq k} (\{Wordlength(y) : y \in Image(v_i)\})。\tag{2.2}$$

この値は小さければ小さいほどよい。これを改善するには、パラメータnをより大きな値に設定するか、異なるベースまたはベース順序を使用してアルゴリズムを再実行すればよい。

表 1. SGSWordQuick の実行時間。

グループ	学位	オーダー	|B|	n	時間 (s)	最大言葉の長さ
PGL3(8)	73	1.6×10^7	4	10^4	88	48
				3×10^4	285	46
ルービックキューブ	48	4.3×10^{19}	18	10^4	110	165
				3×10^4	276	155
				10^6	7289	144
キューブグレー$_5$	32	2.1×10^{26}	28	10^4	124	415
				3×10^4	312	343
				10^6	6643	249
キューブグレー$_6$	64	3.4×10^{70}	60	3×10^4	475	1988
				10^6	8965	936
キューブグレー$_7$	128	8.1×10^{177}	124	3×10^5	12807	4893
				10^6	47114	3843
S^*	20	2.4×10^{18}	19	10^4	116	403
S_{20}	20	2.4×10^{18}	19	10^3	11	37
S^*_{50}	50	3×10^{64}	49	10^5	6370	3449
S_{50}	50	3×10^{64}	49	10^4	551	97

3. 結果と結論

このアルゴリズムをテストするために、異なる特徴を持つ数多くのグループに適用した。ここでは、グループCubeGray$_N$ †のファミリーを使用したが、これらのグループは一般に知られていない。しかし、塩基は大きいが、Image (v_i)の単語長はiの成長に対してほぼ安定に保つことができるため、アルゴリズムをテストするには適していた。これらのグループは、対称的な群である。ここでS^*_Nは(2.1)のような生成子を持つn点上の対称群を表す、一方、S^{**}_Nは、$\Omega := \{1,\ldots,N\}$上の順列によって生成される同じグループを示す:

$$S^{**}_N := \langle (1, 2), (1, 3), \ldots, (1, n-1), (1, n) \rangle. \tag{3.1}$$

対称群は、異なる生成子を使用した場合の効果を示すための極端なケースである。群$PGL_3(8)$は、基底が小さいが次数が大きい場合に良い振る舞いを示す。

表1の全てのCPUタイムはSun Sparc IPX上で、GAPプログラミング言語で書かれたプログラムで達成された。この新しいアルゴリズムが、多くの異なる順列群に対してうまく機能することが実証された。MAGMA (Cannon and Bosma (1994)参照)やGAP (Schönert *et al.* (1992)参照)のような群論システムに組み込むことで、その能力をさらに向上させることができる。

† グループCubeGray$_N$ は、N次元超立方体グラフの頂点の並べ替えのグループのサブグループである。すべての頂点は、隣接する頂点の文字列が1ビットだけ異なるような、長さNの一意なビット列でラベル付けできる。このことから、ハミルトン円（すべての頂点に到達する非交差環状経路）は、常にラベルのグレイコードに沿って見つけることができる。これはもち

ろん、ビット列のN桁のいずれかを固定することで定義される2つの（N - 1）次元部分立方体にも当てはまる。グループCubeGray$_N$ は、N次元のそれぞれの(N - 1)次元部分立方体の頂点の2つのグレイコードハミルトン円に沿ったシフトによって生成される順列のグループである。

謝辞

カールスルーエのアルゴリズム・認知システム研究所に在籍していた元指導教官のT. Beth、そしてS. EgnerとA. Nu¨ckelに感謝したい。

参考文献

アトキンソン, M.D. (編) (1984).*計算群理論*.Academic Press.

Butler, G. (1991).*順列群の基本アルゴリズム*.Springer-Verlag：LNCS 559.Cannon, J., Bosma, W. (1994). *マグマ関数ハンドブック*。シドニー大学。

Hofstadter, D. (1985).*メタマジカル・テーマ：心とパターンの本質を探る*.ベーシック・ブックス。

Leon, J.S. (1980).順列の生成によって与えられる群の基底と強生成集合を求めるアルゴリズムについて。*計算の数学*, **35**, 941-974.

Sch¨onert, M. *他* (1992).*GAP：群、アルゴリズム、プログラミング*.Lehrstuhl D fu¨r Mathematik, RWTH Aachen.

Sims, C.C. (1970).順列群の研究における計算法。Leech, J., ed., *Com-putational Problems in Abstract Algebra*, pp.Pergamon.

原文受領：1994年6月7日
1998年3月2日受理